Formalize the Generalization of Classic Problems

All of the equation systems have a similar form

$$\text{Analysis}_\circ(l) = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\text{Analysis}_\bullet(l') \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

and

$$\text{Analysis}_\bullet(l) = f_l(\text{Analysis}_\circ(l))$$

where $B^l \in blocks(S_*)$

where $\bigsqcup$, F, E, ι, and f_l are the parameters.

Formalize the Generalization of Classic Problems

For our problems the parameters ranged over a small set of values

- $\bigsqcup$ is $\bigcap$ or $\bigcup$
- F is $flow(S_*)$ or $flow^R(S_*)$
- E is $\{init(S_*)\}$ or $final(S_*)$
- ι gives the initial/final analysis information
- f_l is the transfer associated with block l

There are several ways we can decompose these parameters. We choose to consider ι and f_l to be problem specific, leaving two dimensions to consider.

Forward versus Backward Problems

Forward Analyses

- F is $flow(S_*)$
- E is $\{init(S_*)\}$
- ι gives the initial analysis information
- $\text{Analysis}_\circ$ holds statement entry information
- $\text{Analysis}_\bullet$ holds statement exit information

Forward versus Backward Problems

Backward Analyses

- F is $flow^R(S_*)$
- E is $final(S_*)$
- ι gives the final analysis information
- Analysis$_\circ$ holds statement exit information
- Analysis$_\bullet$ holds statement entry information

Greatest versus Least Solution Problems

Greatest Solution

- $\bigsqcup$ is $\bigcap$
- initial value, $\perp$, is set of all data flow facts
- aka *must* analyses, *all paths*/*universal* analysis

Least Solution

- $\bigsqcup$ is $\bigcup$
- initial value, $\perp$, is $\emptyset$
- aka *may* analyses, *existential* analysis

Definition of Data Flow Frameworks

Data flow facts are termed a *properties*

The set of data flow facts are termed a *property space* and denoted by L

Typically, L will be a complete lattice

Property sets flowing along different paths are combined according to a *combination operator*, $\bigsqcup : \mathcal{P}(L) \rightarrow L$.

Ascending Chain Condition

An ascending chain in a lattice is a sequence of ordered values from the lattice, $(l_1, \ldots, l_n)$ such that $\forall_{1 \leq i \leq n} : l_i \sqsubseteq l_{i+1}$

We require that all such chains for property spaces eventually stabilize, i.e., $\exists_n : \forall_{n \leq i} : l_n = l_i$

Do the classic examples satisfy this condition?

Alternate Definitions of Data Flow Frameworks

Hecht's book, Marlowe and Ryder's Acta Informatica paper

Relax the need for a complete lattice

- Require only a *join semi-lattice* (i.e., a lattice in which $\sqcup$, $\sqsubseteq$, but not $\sqcap$, etc.)
- Require a $\perp$ element, such that $l \sqcup \perp = l$
- Require the ascending chain condition

Note there is no $\top$ required since the chain condition assures eventual convergence

Flip the lattice over

- Require only a *meet semi-lattice* (i.e., a lattice in which $\sqcap$, $\sqsupseteq$, but not $\sqcup$, etc.)
- Require a $\top$ element, such that $l \sqcup \top = \top$
- Require the *descending* chain condition

Transfer Functions

The set of transfer functions, $f_l : L \to L$ for $l \in \mathsf{Lab}_*$, is contained in the *function space*, $\mathcal{F}$

In addition to the transfer functions $\mathcal{F}$ has the following properties

- $\mathcal{F}$ contains the identity function
- $\mathcal{F}$ is closed under function composition

We usually require monotonicity which is

- sensible as it says that an increase in data flow information at an input cannot produce a decrease in information at the associated output
- necessary for the development terminating flow analysis algorithms

A Monotone Data Flow Framework consists of

A complete lattice, L, satisfying the ascending chain condition

A set, $\mathcal{F}$, of monotone functions from L to L that contains the identity and is closed under composition

This is the most common type of framework, but not the only one that is used in practice.

A Distributive Data Flow Framework consists of

A complete lattice, L, satisfying the ascending chain condition

A set, $\mathcal{F}$, of *distributive* functions from L to L that contains the identity and is closed under composition

Here we mean that the functions distribute over $\sqcup$, i.e.,

$$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

This is stronger than monotonicity which only requires that

$$f(l_1 \sqcup l_2) \sqsupseteq f(l_1) \sqcup f(l_2)$$

and not the other direction.

A Framework Instance

Parameterizes the abstract framework with information about the program to be analyzed.

An instance consists of

- A framework, i.e., L and $\mathcal{F}$
- a finite flow relation, F (e.g., $flow(S_*)$)
- a finite set of extremal labels, E (e.g., $final(S_*)$)
- an extremal value, $\iota \in L$
- a mapping, f, from the labels Lab_*, mentioned in F and E, to functions in $\mathcal{F}$

A Framework Instance Defines a System of Equations

$$\text{Analysis}_\circ(l) = \bigsqcup\{\text{Analysis}_\bullet(l') \mid (l', l) \in F\} \sqcup \iota_E^l$$

where $\iota_E^l = \begin{cases} \iota & \text{if } l \in E \\ \bot & \text{if } l \notin E \end{cases}$ and

$$\text{Analysis}_\bullet(l) = f_l(\text{Analysis}_\circ(l))$$

The Class Problems are Monotone Framework Instances

Consider the Available Expressions problem

- $L = \mathcal{P}(\textsf{AEbox}_*)$
- $\sqsubseteq = \supseteq$
- $\bigsqcup = \bigcap$
- $\perp = \textsf{AEbox}_*$
- $\iota = \emptyset$
- $E = \{init(S_*)\}$
- $F = flow(S_*)$
- $\mathcal{F} = \{f : L \rightarrow L | \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$
- $f_l(l) = (l \setminus \textsf{kill}([B]^l)) \cup \textsf{gen}([B]^l)$ where $[B]^l \in \textsf{blocks}(S_*)$

Available Expressions as a Monotone Framework Instance

Does the lattice, L, satisfy the ascending chain condition?

- Using AEbox_* means we have a finite set of non-trivial syntactic expressions
- Thus the powerset lattice is finite
- All chains are of finite length and they stabilize at $\top = \emptyset$

Does the set of functions, $\mathcal{F}$, have the necessary properties?

Is $\mathcal{F}$ a Monotone Function Space?

Monotonicity

- let $l \sqsubseteq l'$
- it is clear that $(l \setminus l_k) \sqsubseteq (l' \setminus l_k)$ since l_k is constant
- similarly $((l \setminus l_k) \cup l_g) \sqsubseteq ((l' \setminus l_k) \cup l_g)$ since l_g is constant
- thus, $f(l) \sqsubseteq f(l')$ and f is monotone.

Identity

- let $l_k = \emptyset$ and $l_g = \emptyset$
- then $f(l) = ((l \setminus \emptyset) \cup \emptyset) = l$
- so the identity function is in $\mathcal{F}$

Is $\mathcal{F}$ Closed Under Composition?

- let $f(l) = (l \setminus l_k) \cup l_g$ and $f'(l) = (l \setminus l'_k) \cup l'_g$
- then $(f \circ f')(l) = ((l \setminus l'_k) \cup l'_g) \setminus l_k) \cup l_g$
- which is $(((l \setminus l'_k) \setminus l_k) \cup (l'_g \setminus l_k)) \cup l_g$
- which is $(l \setminus (l'_k \cup l_k) \cup ((l'_g \setminus l_k) \cup l_g)$
- let $l''_k = (l'_k \cup l_k)$ and $l''_g = ((l'_g \setminus l_k) \cup l_g)$
- then $(f \circ f')(l) = (l \setminus l''_k) \cup l''_g$ which is in $\mathcal{F}$

Is $\mathcal{F}$ a Distributive Function Space?

We want to prove that $f(l \sqcup l') = f(l) \sqcup f(l')$ where $f \in \mathcal{F}$ and $\sqcup$ can be either $\cup$ or $\cap$

Consider the case for $\sqcup = \cup$

- let f be $f(l) = (l \setminus l_k) \cup l_g$
- consider $f(l \sqcup l') = f(l \cup l') = ((l \cup l') \setminus l_k) \cup l_g$
- which is $((l \setminus l_k) \cup (l' \setminus l_k)) \cup l_g$
- which is $((l \setminus l_k) \cup l_g) \cup ((l' \setminus l_k) \cup l_g)$
- which is $f(l) \cup f(l') = f(l) \sqcup f(l')$

Is $\mathcal{F}$ a Distributive Function Space?

Consider the case for $\sqcup = \cap$

- let f be $f(l) = (l \setminus l_k) \cup l_g$
- consider $f(l \sqcup l') = f(l \cap l') = ((l \cap l') \setminus l_k) \cup l_g$
- which is $((l \setminus l_k) \cap (l' \setminus l_k)) \cup l_g$
- which is $((l \setminus l_k) \cup l_g) \cap ((l' \setminus l_k) \cup l_g)$
- which is $f(l) \cap f(l') = f(l) \sqcup f(l')$

Our Classic Problems are Distributive Data Flow Frameworks

We'll see that distributivity can be exploited to build more efficient solution algorithms and to yield more precise solutions

Not all problems are distributive, for example constant propogation

Constant Propogation Analysis

One of a variety of analyses that approximates the *values* held by variables during program execution.

More specifically

> For each program point, determines whether or not a variable has a single constant value on all executions leading to that point.

Any ideas about how we should define the data flow framework?

Constant Propogation Analysis

Need a way to represent a variables values at a program point

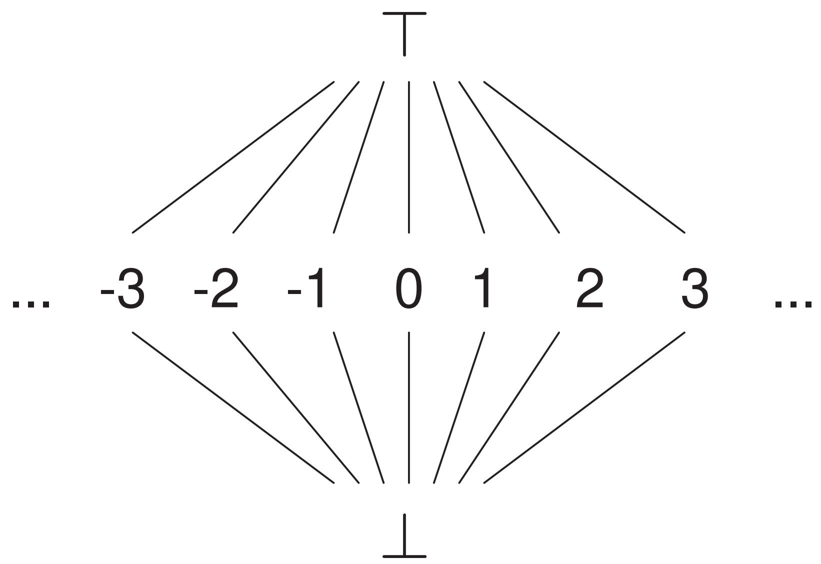
For each variable, we want one approximate value

- use a function f : $\mathbf{Var}_* \rightarrow \mathbf{Z}$

Works if variable is a constant, but how do we represent a non-constant value?

Constant Propogation Analysis

Need a lattice with integer constant values and representation of a non-constant value ($\top$).



$\perp$ denotes no information about the variable value

Constant Propogation Analysis : Complete Lattice (top)

The complete lattice for the problem is

$$L_{\mathsf{CP}} = ((\mathbf{Var}_* \to \mathbf{Z}^\top)_\perp, \sqsubseteq, \sqcup, \sqcap, \perp, \lambda x.\top)$$

where $\mathbf{Z}^\top = \mathbf{Z} \cup \{\top\}$ is ordered as follows

- $\forall z \in \mathbf{Z}^\top : z \sqsubseteq \top$
- $\forall z_1, z_2 \in \mathbf{Z}^\top : (z_1 \sqsubseteq z_2) \Leftrightarrow (z_1 = z_2)$

Conceptually lattice values record the state of an abstract store, $\sigma : \mathbf{Var}_* \to \mathbf{Z}^\top$, at a program point such that

- $\sigma(x) = c$ for $c \in \mathbf{Z}$ means x has constant value c
- $\sigma(x) = \top$ means the value of x is not constant

Constant Propogation Analysis : Complete Lattice (bottom)

The lack of information about *any* variable values is defined by adding an extra value, $\bot$

$(\textbf{Var}_* \rightarrow \textbf{Z}^\top)_\bot$ is ordered as follows

- $\forall \sigma \in (\textbf{Var}_* \rightarrow \textbf{Z}^\top)_\bot : \bot \sqsubseteq \sigma$
- $\forall \sigma_1, \sigma_2 \in (\textbf{Var}_* \rightarrow \textbf{Z}^\top)_\bot : \sigma_1 \sqsubseteq \sigma_2 \Leftrightarrow \forall x : \sigma_1(x) \sqsubseteq \sigma_2(x)$

Any idea why we want this?

Constant Propogation Analysis : Least Upper Bound

The definition of $(\textbf{Var}_* \to \textbf{Z}^\top)_\perp$ is gives us the following lub

- $\forall \sigma \in (\textbf{Var}_* \to \textbf{Z}^\top)_\perp : \sigma \sqcup \perp = \sigma = \perp \sqcup \sigma$
- $\forall \sigma_1, \sigma_2 \in (\textbf{Var}_* \to \textbf{Z}^\top)_\perp : \forall x : (\sigma_1 \sqcup \sigma_2)(x) = \sigma_1(x) \sqcup \sigma_2(x)$

Constant Propogation Analysis : Transfer Functions

For assignment statements we define helper functions to extract *constancy* information from the right-hand sides of type

$$\mathcal{A}_{CP} : \textbf{AExp} \rightarrow (\textbf{Var}_* \rightarrow \textbf{Z}^{\top}_{\perp})$$

where $\textbf{Z}^{\top}_{\perp} = \textbf{Z} \cup \{\top\} \cup \{\perp\}$ with arithmetic operations, $\textbf{op}_a$, lifted as follows:

- $z_1 \widehat{op}_a z_2 = z_1 \textbf{op}_a z_2$ if $z_1, z_2 \in \textbf{Z}$;
- $z_1 \widehat{op}_a z_2 = \perp$ if $z_1 = \perp$ or $z_2 = \perp$; and
- $z_1 \widehat{op}_a z_2 = \top$ otherwise.

Constant Propogation Analysis : Transfer Functions

The helper functions are

$$
\begin{aligned}
\mathcal{A}_{\mathsf{CP}}[\![x]\!](\sigma) &= \begin{cases} \bot & \text{if } \sigma = \bot \\ \sigma(x) & \text{otherwise} \end{cases} \\
\mathcal{A}_{\mathsf{CP}}[\![n]\!](\sigma) &= \begin{cases} \bot & \text{if } \sigma = \bot \\ n & \text{otherwise} \end{cases} \\
\mathcal{A}_{\mathsf{CP}}[\![a_1 op_a a_2]\!](\sigma) &= \mathcal{A}_{\mathsf{CP}}[\![a_1]\!]\sigma \;\; \widehat{op}_a \;\; \mathcal{A}_{\mathsf{CP}}[\![a_2]\!]\sigma
\end{aligned}
$$

Constant Propogation Analysis : Transfer Functions

Now we extract constancy information from the RHS using $\mathcal{A}_{CP}$ to define the transfer function for assignments

- $[x := a]^l : f_l^{\text{CP}}(\sigma) = \begin{cases} \bot & \text{if} \sigma = \bot \\ \sigma[x \mapsto \mathcal{A}_{CP}[\![a]\!]\sigma] & \text{otherwise} \end{cases}$

where $\sigma[x \mapsto n]$ is σ with the mapping for x set to n.

Skips and tests in `While` do not change variable values so we can use an identity transfer function, i.e.,

- $[skip]^l : f_l^{\text{CP}}(\sigma) = \sigma$
- $[b]^l : f_l^{\text{CP}}(\sigma) = \sigma$

Constant Propogation Analysis is an Instance of a Monotone Framework

Is lattice complete?

Is function space monotone?

Parameters?

Constont Propogation as Monotone Framework Instances

- $L == ((\mathbf{Var}_* \to \mathbf{Z}^\top)_\perp$;
- $\sqsubseteq$, defined such that $\forall \sigma_1, \sigma_2 : \sigma_1 \sqsubseteq \sigma_2 \Leftrightarrow \forall x : \sigma_1(x) \sqsubseteq \sigma_2(x)$ (where $\forall x : x \sqsubseteq \top$), and $\forall \sigma : \perp \sqsubseteq \sigma$;
- $\bigsqcup$ meets the mapped values for all variables in the underlying $\mathbf{Z}^\top$ poset;
- $\perp$ is a special token added to represent lack of any variable value information;
- $\iota = \lambda x.\top$ representing an unknown initial value;
- $E = \{init(S_*)\}$;
- $F = flow(S_*)$; and
- $\mathcal{F}$ as defined above for the three kinds of statements.

Constant Propogation as a Monotone Framework Instance

Does the lattice, L, satisfy the ascending chain condition?

- Height of lattice is 3!

Does the set of functions, $\mathcal{F}$, have the necessary properties?

Monotonicity and Closure (by definition)

Identity (explicitly included for skip and branches)

Constant Propogation Analysis is **not** an Instance of a Distributive Framework

Consider f_l^{CP} for $[y := x * x]^l$

- let σ_1 be such that $\sigma_1(x) = 1$
- let σ_2 be such that $\sigma_2(x) = -1$
- by definition $\forall x : (\sigma_1 \sqcup \sigma_2)(x) = \sigma_1(x) \sqcup \sigma_2(x)$
- so $(\sigma_1 \sqcup \sigma_2)$ maps x to $\top$ and $f_l^{\mathsf{CP}}(\sigma_1 \sqcup \sigma_2)$ maps y to $\top$
- but y has the value 1 at l
- which is captured by $f_l^{\mathsf{CP}}(\sigma_1)$ and $f_l^{\mathsf{CP}}(\sigma_2)$, i.e., both yield value 1 for y
- and $f_l^{\mathsf{CP}}(\sigma_1) \sqcup f_l^{\mathsf{CP}}(\sigma_2)$ maps y to 1

Since $f_l^{\mathsf{CP}}(\sigma_1 \sqcup \sigma_2) \neq f_l^{\mathsf{CP}}(\sigma_1) \sqcup f_l^{\mathsf{CP}}(\sigma_2)$ the problem is not distributive.